Laravel Reverb

Introduction

Laravel Reverb brings blazing-fast and scalable real-time WebSocket communication directly to your Laravel application, and provides seamless integration with Laravel's existing suite of event broadcasting tools.

Installation

You may install Reverb using the `install:broadcasting` Artisan command:

```
php artisan install:broadcasting
```

Configuration

Behind the scenes, the `install:broadcasting` Artisan command will run the `reverb:install` command, which will install Reverb with a sensible set of default configuration options. If you would like to make any configuration changes, you may do so by updating Reverb's environment variables or by updating the `config/reverb.php` configuration file.

Application Credentials

In order to establish a connection to Reverb, a set of Reverb "application" credentials must be exchanged between the client and server. These credentials are configured on the server and are used to verify the request from the client. You may define these credentials using the following environment variables:

```
REVERB_APP_ID=my-app-id
REVERB_APP_KEY=my-app-key
REVERB_APP_SECRET=my-app-secret
```

Allowed Origins

You may also define the origins from which client requests may originate by updating the value of the `allowed_origins` configuration value within the `apps` section of the `config/reverb.php` configuration file. Any requests from an origin not listed in your allowed origins will be rejected. You may allow all origins using `*`:

```
'apps' => [
    [
        'app_id' => 'my-app-id',
        'allowed_origins' => ['laravel.com'],
        // ...
    ]
]
```

Additional Applications

Typically, Reverb provides a WebSocket server for the application in which it is installed. However, it is possible to serve more than one application using a single Reverb

installation.

For example, you may wish to maintain a single Laravel application which, via Reverb, provides WebSocket connectivity for multiple applications. This can be achieved by defining multiple `apps` in your application's `config/reverb.php` configuration file:

```
'apps' => [
    [
        'app_id' => 'my-app-one',
        // ...
    ],
    [
        'app_id' => 'my-app-two',
        // ...
    ],
],
```

SSL

In most cases, secure WebSocket connections are handled by the upstream web server (Nginx, etc.) before the request is proxied to your Reverb server.

However, it can sometimes be useful, such as during local development, for the Reverb server to handle secure connections directly. If you are using Laravel Herd's secure site feature or you are using Laravel Valet and have run the secure command against your application, you may use the Herd / Valet certificate generated for your site to secure your Reverb connections. To do so, set the `REVERB_HOST` environment variable to your site's hostname or explicitly pass the hostname option when starting the Reverb server:

```
php artisan reverb:start --host="0.0.0.0" --port=8080 --hostname="laravel.tes
```

Since Herd and Valet domains resolve to `localhost`, running the command above will result in your Reverb server being accessible via the secure WebSocket protocol (`wss`) at `wss://laravel.test:8080`.

You may also manually choose a certificate by defining `tls` options in your application's `config/reverb.php` configuration file. Within the array of `tls` options, you may provide any of the options supported by PHP's SSL context options:

```
'options' => [
    'tls' => [
        'local_cert' => '/path/to/cert.pem'
```

```
        ],
    ],
```

Running the Server

The Reverb server can be started using the `reverb:start` Artisan command:

```
php artisan reverb:start
```

By default, the Reverb server will be started at `0.0.0.0:8080`, making it accessible from all network interfaces.

If you need to specify a custom host or port, you may do so via the `--host` and `--port` options when starting the server:

```
php artisan reverb:start --host=127.0.0.1 --port=9000
```

Alternatively, you may define `REVERB_SERVER_HOST` and `REVERB_SERVER_PORT` environment variables in your application's `.env` configuration file.

The `REVERB_SERVER_HOST` and `REVERB_SERVER_PORT` environment variables should not be confused with `REVERB_HOST` and `REVERB_PORT`. The former specify the host and port on which to run the Reverb server itself, while the latter pair instruct Laravel where to send broadcast messages. For example, in a production environment, you may route requests from your public Reverb hostname on port `443` to a Reverb server operating on `0.0.0.0:8080`. In this scenario, your environment variables would be defined as follows:

```
REVERB_SERVER_HOST=0.0.0.0
REVERB_SERVER_PORT=8080

REVERB_HOST=ws.laravel.com
REVERB_PORT=443
```

Debugging

To improve performance, Reverb does not output any debug information by default. If you would like to see the stream of data passing through your Reverb server, you may provide the `--debug` option to the `reverb:start` command:

```
php artisan reverb:start --debug
```

Restarting

Since Reverb is a long-running process, changes to your code will not be reflected without restarting the server via the `reverb:restart` Artisan command.

The `reverb:restart` command ensures all connections are gracefully terminated before stopping the server. If you are running Reverb with a process manager such as Supervisor, the server will be automatically restarted by the process manager after all connections have been terminated:

```
php artisan reverb:restart
```

Monitoring

Reverb may be monitored via an integration with Laravel Pulse. By enabling Reverb's Pulse integration, you may track the number of connections and messages being handled by your server.

To enable the integration, you should first ensure you have installed Pulse. Then, add any of Reverb's recorders to your application's `config/pulse.php` configuration file:

```
use Laravel\Reverb\Pulse\Recorders\ReverbConnections;
use Laravel\Reverb\Pulse\Recorders\ReverbMessages;

'recorders' => [
    ReverbConnections::class => [
        'sample_rate' => 1,
    ],

    ReverbMessages::class => [
        'sample_rate' => 1,
    ],

    // ...
],
```

Next, add the Pulse cards for each recorder to your Pulse dashboard:

```
<x-pulse>
    <livewire:reverb.connections cols="full" />
    <livewire:reverb.messages cols="full" />
    ...
</x-pulse>
```

Connection activity is recorded by polling for new updates on a periodic basis. To ensure this information is rendered correctly on the Pulse dashboard, you must run the `pulse:check` daemon on your Reverb server. If you are running Reverb in a horizontally scaled configuration, you should only run this daemon on one of your servers.

Running Reverb in Production

Due to the long-running nature of WebSocket servers, you may need to make some optimizations to your server and hosting environment to ensure your Reverb server can effectively handle the optimal number of connections for the resources available on your server.

Laravel Cloud offers fully managed WebSocket infrastructure powered by Laravel Reverb clusters, allowing you to scale and ship Reverb enabled applications without managing infrastructure.

Open Files

Each WebSocket connection is held in memory until either the client or server disconnects. In Unix and Unix-like environments, each connection is represented by a file. However, there are often limits on the number of allowed open files at both the operating system and application level.

Operating System

On a Unix based operating system, you may determine the allowed number of open files using the `ulimit` command:

```
ulimit -n
```

This command will display the open file limits allowed for different users. You may update these values by editing the `/etc/security/limits.conf` file. For example, updating the maximum number of open files to 10,000 for the `forge` user would look like the following:

```
# /etc/security/limits.conf
forge        soft  nofile  10000
forge        hard  nofile  10000
```

Event Loop

Under the hood, Reverb uses a ReactPHP event loop to manage WebSocket connections on the server. By default, this event loop is powered by `stream_select`, which doesn't require any additional extensions. However, `stream_select` is typically limited to 1,024 open files. As such, if you plan to handle more than 1,000 concurrent connections, you will need to use an alternative event loop not bound to the same restrictions.

Reverb will automatically switch to an `ext-uv` powered loop when available. This PHP extension is available for install via PECL:

```
pecl install uv
```

Web Server

In most cases, Reverb runs on a non web-facing port on your server. So, in order to route traffic to Reverb, you should configure a reverse proxy. Assuming Reverb is running on host `0.0.0.0` and port `8080` and your server utilizes the Nginx web server, a reverse proxy can be defined for your Reverb server using the following Nginx site configuration:

```
server {
    ...

    location / {
        proxy_http_version 1.1;
        proxy_set_header Host $http_host;
        proxy_set_header Scheme $scheme;
        proxy_set_header SERVER_PORT $server_port;
        proxy_set_header REMOTE_ADDR $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";

        proxy_pass http://0.0.0.0:8080;
    }

    ...
}
```

Reverb listens for WebSocket connections at `/app` and handles API requests at `/apps`. You should ensure the web server handling Reverb requests can serve both of these URIs. If you are using Laravel Forge to manage your servers, your Reverb server will be correctly configured by default.

Typically, web servers are configured to limit the number of allowed connections in order to prevent overloading the server. To increase the number of allowed connections on an Nginx web server to 10,000, the `worker_rlimit_nofile` and `worker_connections` values of the `nginx.conf` file should be updated:

```
user forge;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
worker_rlimit_nofile 10000;

events {
  worker_connections 10000;
  multi_accept on;
}
```

The configuration above will allow up to 10,000 Nginx workers per process to be spawned. In addition, this configuration sets Nginx's open file limit to 10,000.

Ports

Unix-based operating systems typically limit the number of ports which can be opened on the server. You may see the current allowed range via the following command:

```
cat /proc/sys/net/ipv4/ip_local_port_range
# 32768    60999
```

The output above shows the server can handle a maximum of 28,231 (60,999 - 32,768) connections since each connection requires a free port. Although we recommend horizontal scaling to increase the number of allowed connections, you may increase the number of available open ports by updating the allowed port range in your server's `/etc/sysctl.conf` configuration file.

Process Management

In most cases, you should use a process manager such as Supervisor to ensure the Reverb server is continually running. If you are using Supervisor to run Reverb, you should update the `minfds` setting of your server's `supervisor.conf` file to ensure Supervisor is able to open the files required to handle connections to your Reverb server:

```
[supervisord]
...
minfds=10000
```

Scaling

If you need to handle more connections than a single server will allow, you may scale your Reverb server horizontally. Utilizing the publish / subscribe capabilities of Redis, Reverb is able to manage connections across multiple servers. When a message is received by one of your application's Reverb servers, the server will use Redis to publish the incoming message to all other servers.

To enable horizontal scaling, you should set the `REVERB_SCALING_ENABLED` environment variable to `true` in your application's `.env` configuration file:

```
REVERB_SCALING_ENABLED=true
```

Next, you should have a dedicated, central Redis server to which all of the Reverb servers will communicate. Reverb will use the default Redis connection configured for your application to publish messages to all of your Reverb servers.

Once you have enabled Reverb's scaling option and configured a Redis server, you may simply invoke the `reverb:start` command on multiple servers that are able to communicate with your Redis server. These Reverb servers should be placed behind a load balancer that distributes incoming requests evenly among the servers.

Events

Reverb dispatches internal events during the lifecycle of a connection and message handling. You may listen for these events to perform actions when connections are managed or messages are exchanged.

The following events are dispatched by Reverb:

```
Laravel\Reverb\Events\ChannelCreated
```

Dispatched when a channel is created. This typically occurs when the first connection subscribes to a specific channel. The event receives the `Laravel\Reverb\Protocols\Pusher\Channel` instance.

`Laravel\Reverb\Events\ChannelRemoved`

Dispatched when a channel is removed. This typically occurs when the last connection unsubscribes from a channel. The event receives the `Laravel\Reverb\Protocols\Pusher\Channel` instance.

`Laravel\Reverb\Events\ConnectionPruned`

Dispatched when a stale connection is pruned by the server. The event receives the `Laravel\Reverb\Contracts\Connection` instance.

`Laravel\Reverb\Events\MessageReceived`

Dispatched when a message is received from a client connection. The event receives the `Laravel\Reverb\Contracts\Connection` instance and the raw string `$message`.

`Laravel\Reverb\Events\MessageSent`

Dispatched when a message is sent to a client connection. The event receives the `Laravel\Reverb\Contracts\Connection` instance and the raw string `$message`.

Pint

Sail

Sanctum

Socialite

Telescope

Pulse

Reverb

Echo

Resources

Documentation

Starter Kits

Release Notes

Blog

News

Community

Larabelles

Learn

Jobs

Careers

Trust

Partners

Jump24

Kirschbaum

Active Logic

Redberry

Tighten

byte5

Curotec

VehikI

More Partners


Laravel